Social Media Dashboard

IIA Deadline-2

Data acquisition and sources

Himanshu-2021464
Siddharth-2021493
Shivam-2021489

DELIVERIES

1. **What data/open tools/software is required?**

    - Data - Analytical user data from multiple sources(YouTube, twitch, and Dailymotion).
    - Open Tools - Apache Hadoop for ETL.
    - Other software - Google Cloud for Youtube API, Pycharm, Google Colab, and Google Sheets for running the script to extract the data from multiple sources and storing them in a .csv file.

2. **For data simulation, we use the API's services provided by the developers of multiple platforms.**

    - YouTube Data API v3 and YouTube Analytics API for getting the YouTube analytics data of users.

    - Dailymotion API for getting analytical data for Dailymotion users.

      1) Register your application on the Dailymotion Developer Portal.
      2) Obtain the necessary API credentials (client ID and client secret)
      3) Authenticate your application using OAuth 2.0.
      4) Make API requests to retrieve the desired analytical data.

      Dailymotion's API documentation provides detailed information on endpoints, query parameters, authentication, and examples of how to use the API to retrieve analytics data. Always refer to the official

documentation for the most accurate and up-to-date information on using the Dailymotion API for analytics.

- Twitch API for getting analytical data for Twitch streamers/users.

To get these data using the APIs, we use a Python script to run all the necessary commands, get the data, and store the data using API after authentication by the API KEYs.

3. **Global and local schema for data sources.**

   Global Schema:
   (Platform, Title, View, Likes, Dislikes, Comments, Duration, Tags, Published_Date, Language, URL, Reaction)

   Local Schema:

   There are three sources for Local Schema: YouTube, Twitch, and Dailymotion.

   - YouTube Local Schema:
     youtube(title, description, published, tag_count, view_count, like_count, dislike_count, comment_count,title_length,reaction)

   - Twitch Local Schema:

     twitch(broadcaster_id broadcaster_name ,followers,game_name, video_id ,URL, language,view_count,created_at, duration,tags, title)

   - Dailymotion(title,views_total,likes_total,rating,duration,tags,published_date ,audience_total)

The global schema includes standard fields that are shared across all data sources (Platform, Title, Views, Likes, Dislikes, Comments, Duration, Tags, Published_Date, Language, URL, and Reaction), allowing for unified analysis and comparison of data from different platforms. The local schemas specify the fields unique to each data source, making it clear which fields originate from YouTube, Twitch, or Dailymotion. This organization helps maintain data separation while providing a consistent structure for analysis.

API Access:

We are using the APIs to retrieve data Programmatically. The API support for each local source is different.
We need to authenticate and make API requests to fetch data for all these APIs by generating and using API KEY, authentication token, and client ID.

Communications through API:

To communicate with local sources through APIs from a global source (virtual or materialized information integration), we would typically use API endpoints provided by each local source. This may involve:

- Authenticating with the local source's API using API keys, OAuth tokens, or other authentication methods.
- Constructing API requests to fetch the required data.
- Handling API responses may be in JSON, XML, or other formats.
- Aggregating and transforming data retrieved from multiple local sources as needed.

Communication between Global and Local Sources:

- Communication between global and local sources can be facilitated through middleware, integration platforms, or custom-built connectors. These components act as intermediaries to route requests and responses between the global and local sources.

Some considerations for communication include the following:

- Ensuring data security and access control during communication.
- Managing data synchronization to keep global and local sources up-to-date.
- Handling errors and retries in case of failed communication.
- Monitoring and logging communication activities for auditing and troubleshooting.

Federated Search Query on Local Sources:

In scenarios where we need to perform federated searches across local sources, we might include specifying which local sources to query, filtering criteria, date ranges, and other parameters.

For example, we might want to search for social media posts mentioning a specific keyword ("search criteria") within the last 30 days ("constraints") across multiple social media platforms. Federated search queries would be executed against the respective local sources, and results would be aggregated and presented globally.

#sample CODE SNIPPET for Youtube data simulation.

```python
def get_video_details(youtube, video_list):
    stats_list = []

    for i in range(0, len(video_list), 50):
        request = youtube.videos().list(
            part="snippet,contentDetails,statistics",
            id=video_list[i:i + 50]
        )

        data = request.execute()
        for video in data['items']:
            title = video['snippet']['title']
            published = video['snippet']['publishedAt']
            description = video['snippet']['description']

            # Check if 'tags' field exists and handle it gracefully
            if 'tags' in video['snippet']:
                tags = video['snippet']['tags']
                tag_count = len(tags)
            else:
                tags = []
                tag_count = 0

            view_count = video['statistics'].get('viewCount', 0)
            like_count = video['statistics'].get('likeCount', 0)
            dislike_count = video['statistics'].get('dislikeCount', 0)
            comment_count = video['statistics'].get('commentCount', 0)
            stats_dict = dict(
                title=title,
                description=description,
                published=published,
                tag_count=tag_count,
                view_count=view_count,
                like_count=like_count,
                dislike_count=dislike_count,
                comment_count=comment_count
            )
            stats_list.append(stats_dict)

    return stats_list
```

#sample CODE SNIPPET for fetching Twitch follower count using API's.

```python
import requests
import json
#Getting followers of a user-----------------------------------------------------
# Defining the API endpoint and headers
url = 'https://api.twitch.tv/helix/channels/followers'
params = {'broadcaster_id': '37402112'}
headers = {
    'Authorization': 'Bearer sgmzzxu0ruk860hex7fcc8a6z37exe',
    'Client-Id': 'emf8vrnqgh7qj5g3fj1tcq6izpl8r8',
}

# Making the API request
response = requests.get(url, params=params, headers=headers)

# Checking if the request was successful
if response.status_code == 200:
    data = response.json()

    # Define the file name
    get_followers = 'followers.json'

    # Writing the raw JSON data to a JSON file
    with open(get_followers, 'w', encoding='utf-8') as jsonfile:
        json.dump(data, jsonfile, ensure_ascii=False, indent=4)

    print(f'Raw JSON data has been saved to {get_followers}')
else:
    print(f'Error: {response.status_code} - {response.text}')
#------------------------------------------------------------------------------
```

References:

1. https://dev.twitch.tv/docs/api/reference/
2. https://dev.twitch.tv/docs/api/
3. https://console.cloud.google.com/
4. https://developers.dailymotion.com/api/
5. https://developers.google.com/youtube/analytics
6. https://developers.google.com/youtube/v3